Distributed Systems - Security

Foundations, Covert Channels, Non Interference

Marcus Völp / Hermann Härtig

2008

TECHNISCHE
UNIVERSITÄT
DRESDEN

Purpose of this Lecture

- Some selected formal methods in security
  - Formal / precise definition of security properties
  - Proving security properties

- Security Evaluation
  - Common Criteria EAL 7 / A1 and beyond
  - German Information Security Agency (GISA) Q7

Purpose of this Lecture

- GISA IT Security Evaluation Criteria (Q7)
  - "The machine language of the processor used shall to a great extent be formally defined."
  - "The consistency between the lowest specification level and the source code shall be formally verified."#
  - "The source code will be examined for the existence of covert channels, applying formal methods. It will be checked that all covert channels detected which cannot be eliminated are documented. [...]"

Overview

- Introduction
- Safety Question
  - Decidability and Protection Models
- Security Policies
  - Policy Enforcement
- Enforcement of Information Flow Policies by Static Code Analysis
  - Noninterference
  - Security Type Systems

Introduction: Security Policies

- Definition:
  - A *security policy* is a statement that partitions the states of the system into a set of authorized, or secure, states and a set of unauthorized, or nonsecure, states.
  - A *secure system* is a system that starts in an authorized state and cannot enter an unauthorized state.
- Example:
  - Policy: only root and I are allowed to read foo.txt
  - Enforcement: foo.txt u+r (g,a -r)
  - Secure system? No – owner can change rights to a+r

Introduction: Confidentiality, Integrity, Availability

- Confidentiality:
  - Prevent unauthorized disclosure of information

  Definition 1a: *Information I is* ***confidential*** *with respect to set of entities X if no member of X can obtain information about I.*

  Definition 1b: *Only authorized users (entities, principals, etc.) can access information (data, programs, etc.)*

Introduction: Confidentiality, Integrity, Availability

- Integrity:
  - Correctness of data and information (trust)

  *<u>Definition 2a:</u> Information I is **integer** with respect to X if all members of X trust I.*

  <u>Definition 2b:</u> Either information is current, correct, and complete, or it is possible to detect that these properties do not hold.

- Recoverability:

  <u>Definition 3b:</u> Information that has been damaged can be recovered eventually.

Introduction: Confidentiality, Integrity, Availability

- Availability:
  - Accessibility of information and services

  <u>*Definition 4a:*</u> *Resource I is **available** with respect to X if all members of X can access I.*

  <u>*Definition 4b:*</u> *Data is **available** when and where an authorized user needs it.*

Introduction: Access Control Matrix

Subjects \ Objects	File 1	File 2	Process1	Process2
Process1	read, write	read	read, write, execute	write
Process2	read	read	read	read, write, execute

- Protection State Transitions:
  - $X_i \mid\!-_{t_{i+1}} X_{i+1}$ States X_j , Commands t_k
  - $X \mid\!- {}^*Y$ Sequence
  - Access Control Matrix: (S, O, P) with Subjects S, Objects O and Permissions P

Introduction: Access Control Matrix

- Commands
  - **create subject s**

    Pre: $s \notin S$,

    Post: $S' = S \cup \{s\}$, $O' = O \cup \{s\}$,
    $\forall x \in O': p'(s, x) = \emptyset$, $\forall y \in S': p'(y, s) = \emptyset$,
    $\forall x \in O, y \in S : p'(x, y) = p(x, y)$

  - **enter r into p(s,o)**

    Pre: $s \in S$, $o \in O$

    Post: $S' = S$, $O' = O$,

    $\forall x \in O', y \in S': (s,o) \neq (x, y) \Rightarrow p'(x,y) = p(x, y)$

    $p'(s, o) = p(s, o) \cup \{r\}$

Introduction: Access Control Matrix

- Further operations:
  - create object o
  - delete right r from p(s,o)
  - destroy subject s
  - destroy object o

Principle of Attenuation

- A subject may not give rights it does not possess to another.

  - **enter r into p(s,o)**

    Pre: $s \in S$, $o \in O$

    Post: $S' = S$, $O' = O$,

    $\forall\, x \in O', y \in S': (s,o) \neq (x, y) \Rightarrow p'(x,y) = p(x, y)$

    $p'(s, o) = p(s, o) \cup \{r\}$

  - **f.grant r into p(s,o)**

    if r **in** p(f,o) **then**

    enter r **into** p(s,o)

Safety Question

- **<u>Definition: Leakage</u>**
  When a right r is added to an element of the ACM not already containing r, r is said to be *leaked.*

- Is the system *safe with respect to right r*, i.e., can it never happen that the system (including s_0) leaks the right r?

- **<u>Safety Question:</u>**
  Is there an algorithm for determining whether a given protection system with initial state s_0 is safe with respect to r?

Safety Question: Decidability

- **Theorem:**
  It is undecidable whether a given state of a given protection system is safe for a given generic right.

- **Proof by contradiction:**
  Reduction of the halting problem of an arbitrary Turing machine to the safety problem. (next slide)

- However, safety is decidable systems with more specific rules:
  - Monoconditional (only one condition in if clause) monotonic (no destroy command) systems.
  - Take-Grant protection model

Safety Question: Decidability

- Proof Sketch:
  - Turing Machine: T (tape symbols M, states K, δ)
    - δ: K x M -> K x M x {L,R}

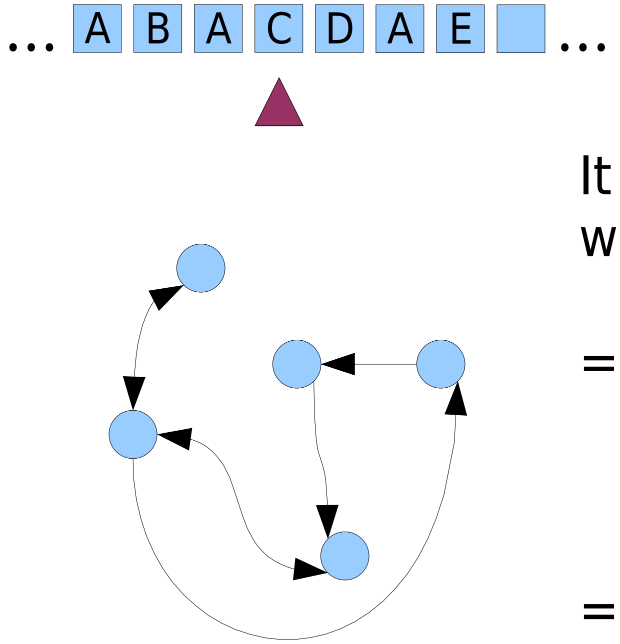
      e.g., δ: (x, A) -> (y, B, L)

  - "Implement Turing Machine with ACM"
    - states, symbols -> generic rights
    - cell i -> subject s_i
    - Head:

      head in cell j, T in state x => $x \in p(s_j, s_j)$

Turing Machine

- http://wiki...



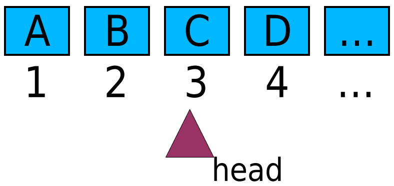
It is undecideable whether the TM will halt given an arbitrary program

=> if S is an implementation of the TM then S can be used to execute the program given to the TM

=> whether S will halt is undecidable for general programs

Safety Question: Decidability

- Proof Sketch:



	s_1	s_2	s_3	s_4
s_1	A	own		
s_2		B	own	
s_3			C,*x*	own
s_4				D,*end*

- Command **δ**: (*x*, *A*) -> (*y*, *B*, L)
  if *own* in $p(s_{i-1}, s_i)$ and x in $p(s_i, s_i)$ and A in $p(s_i, s_i)$ then
  delete *x* from $p(s_i, s_i)$
  delete *A* from $p(s_i, s_i)$
  enter *B* into $p(s_i, s_i)$
  enter *y* into $p(s_{i-1}, s_{i-1})$

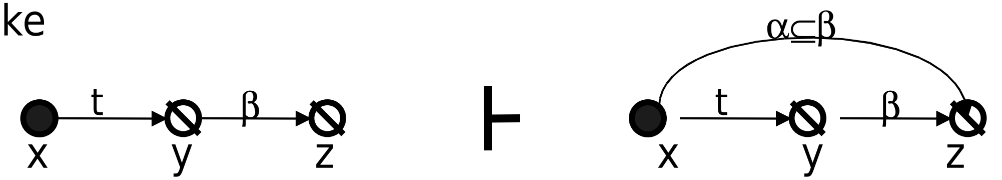
  - Similar commands for other **δ**

If Turing machine enters state q_f then the protection system has leaked right q_f ; otherwise the protection system is safe for generic right . But whether T enters the (halting) state q_f is undecidable.

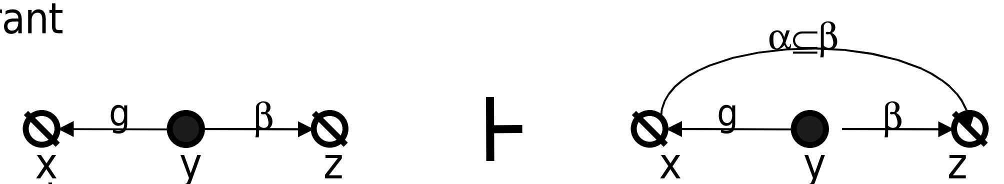
Take-Grant Protection Model

- Directed Graph
  - Vertices: object, subject (either object or subject)
  - Edges: r subject has right r on object

  - Transition Rules:
    - Take



    - Grant

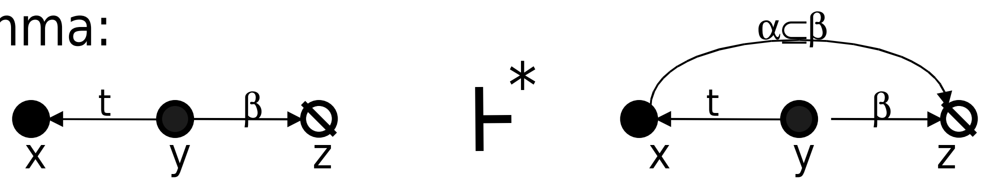


    - Create

    - Remove

Take-Grant Protection Model

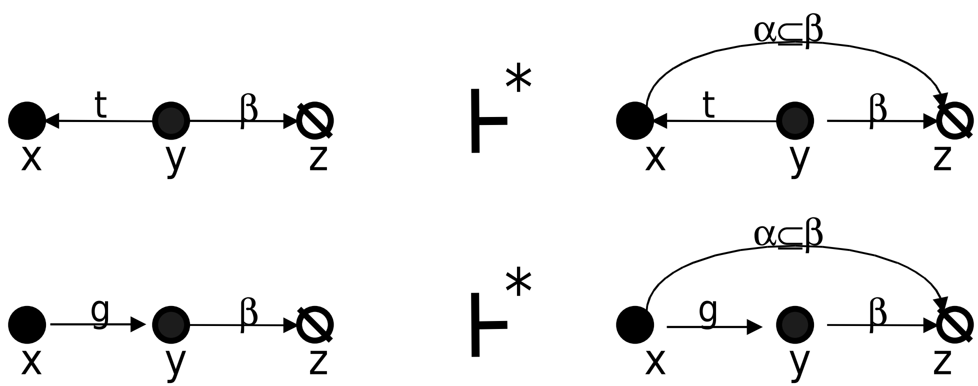
- Sharing and Thiefs
  - can share (α, x, z, G_0)
  - Lemma:



  - Proof:
    x.create v (tg) ; y.take g ; y.grant α to v ; x.take α from v

Take-Grant Protection Model

- Safety is decidable in Take-Grant
  - Proof Sketch:
    - transition rules + lemmas allows generation of graph showing potential access



    - generate potential access graph
    - reason about safety in potential access graph directly
  - Remark: looking at the current system suffices (safety is decidable in linear time)

Summary

- Security is concerned with
  - Confidentiality
  - Integrity
  - Avilability

- Safety
  - In general not decidable
  - Undecidable for unrestricted Access Control Matrix

  - There are decidable protection models
    (e.g., Take-Grant Capability Model)

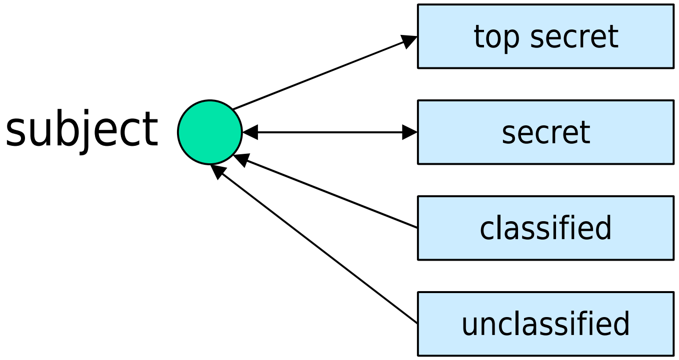
Overview

- Introduction
- Safety Question
  - Decidability and Protection Models
- Security Policies
  - Policy Enforcement
- Information Flow
  - Covert Channels
    - Definition
    - Detection
  - Non Interference and Unwinding Theorems

Security Policies

- Classification
  - Concern:
    - Confidentiality Policies e.g., Bell La Padula
    - Integrity Policies e.g., Biba, (Inventory System)
    - Availability Policies
    - Hybrid e.g., Chinese Wall, (Clinical Information System)
  - Discretionary
    - User can set access control mechanism to allow or deny access to an object.
  - Mandatory
    - System mechanism controls access to an object; individual users cannot alter this access.

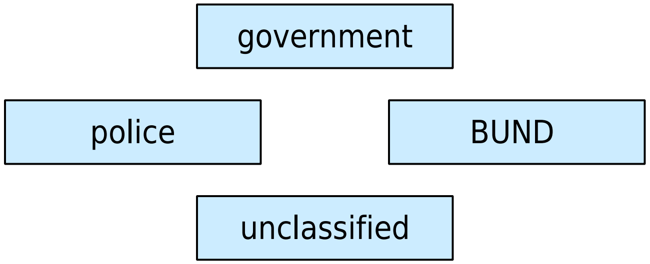
Multi Level Security



Relation $\leq$: L x L defines **total order** of labels

- *-property *(who can write?)*
  - S can write O if and only if Label(S) <= Label(O)
- basic security condition *(who can read?)*
  - S can read O if and only if Label(O) <= Label(S)

Lattice [D.Denning '76]



- Relation ≤ defines **partial order** of security levels
- Least upper bound exists for any finite subset

Confidentiality: L ≤ H

Integrity: h ≤ l

Low-Water-Mark / Biba Integrity Policy

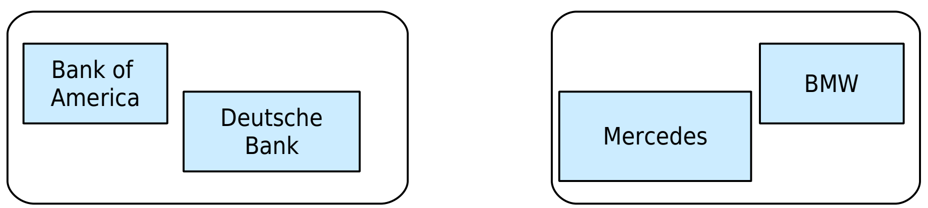
- Integrity Labels similar to secrecy labels:
  - Idea: Data produced by source of varying *trusted.*
  - Using less trusted data will influence the results

- Low Water Mark
  - s can write to o if and only if I(o) <= I(s)
  - If s reads o then I'(s) = min(I(s), I(o))
  - s_1 can execute s_2 if and only if $I(s_2) <= I(s_1)$
    - Problem: decrease of integrity level
- Biba
  - s can read o if and only if I(s) <= I(o)
  - s can write o if and only if I(o) <= I(s)
  - s_1 can execute s_2 if and only if $I(s_2) <= I(s_1)$

Chinese Wall

- Conflict of Interest
  - British law e.g., in stock exchange
    - Trader represents two clients and best interest of clients conflict (trader could help one gain at expense of other)

Conflict of interrest classes



  - Simple Security
    - S can read O iff
      - ∃ O' accessed by S with CD(O') = CD(O), or,
      - ∀ O' read by S => COI(O') ≠ COI(O)

Company Dataset (→ CD)

  - * property
    - S may write O iff
      - S can read O, and,
      - Forall O' readable by S => CD(O') = CD(O)

Policy Enforcement Mechanisms

- Access Control List (classical)
  - OS keeps list of processes x rights for each object
  - acl(file1) = { (process 1, {read, write, execute}), (process 2 {read})}
  - acl(process1) = {(process 1, {read, write, execute})}
  - acl(process2) = {(process 1, {write}), (process 2, {r, w, x})}

- Abbreviations:
  - Groups: Unix, AIX
  - Wildcards:
    - p, *, read (read access to p regardless in which group p is)
- Conflicts:
  - two opposing rights in ACL (group +r, user –r)
    - order of occurance in ACL: Cisco Router
    - deny > allow: AIX

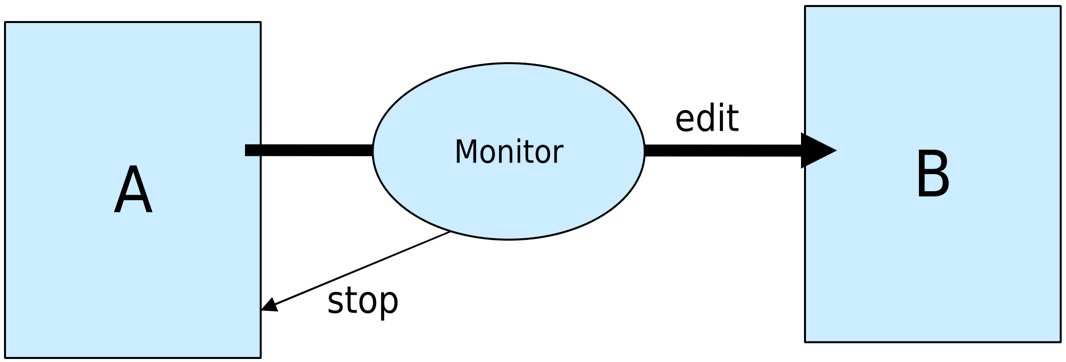
- Problems: modification

Policy Enforcement Mechanisms

- Capabilities
  - caps(process 1) = {(file1, {read, write}), (file2, {read})}
- Implementation:
  - Store capabilities in per process segment / page protected by kernel (e.g. page permission = supervisor) (e.g., CAP)
  - Cryptography (e.g., Amoeba)
  - Hardware tags associated with each word (rarely used e.g., B5700)
- Copying:
  - Take, grant permissions on capabilities
  - Copy flag
- Revocation:
  - Local:
    - Linked list / Tree (e.g., Mapping Database) of all capabilities
    - Indirection: Object which stores capabilities,
indirection right authorizes use but not take or grant of capability
revoke by destroying indirection object
  - Remote:
    - Expiry information

Policy Enforcement Mechanisms

- Monitoring: (Schneider / Bauer)

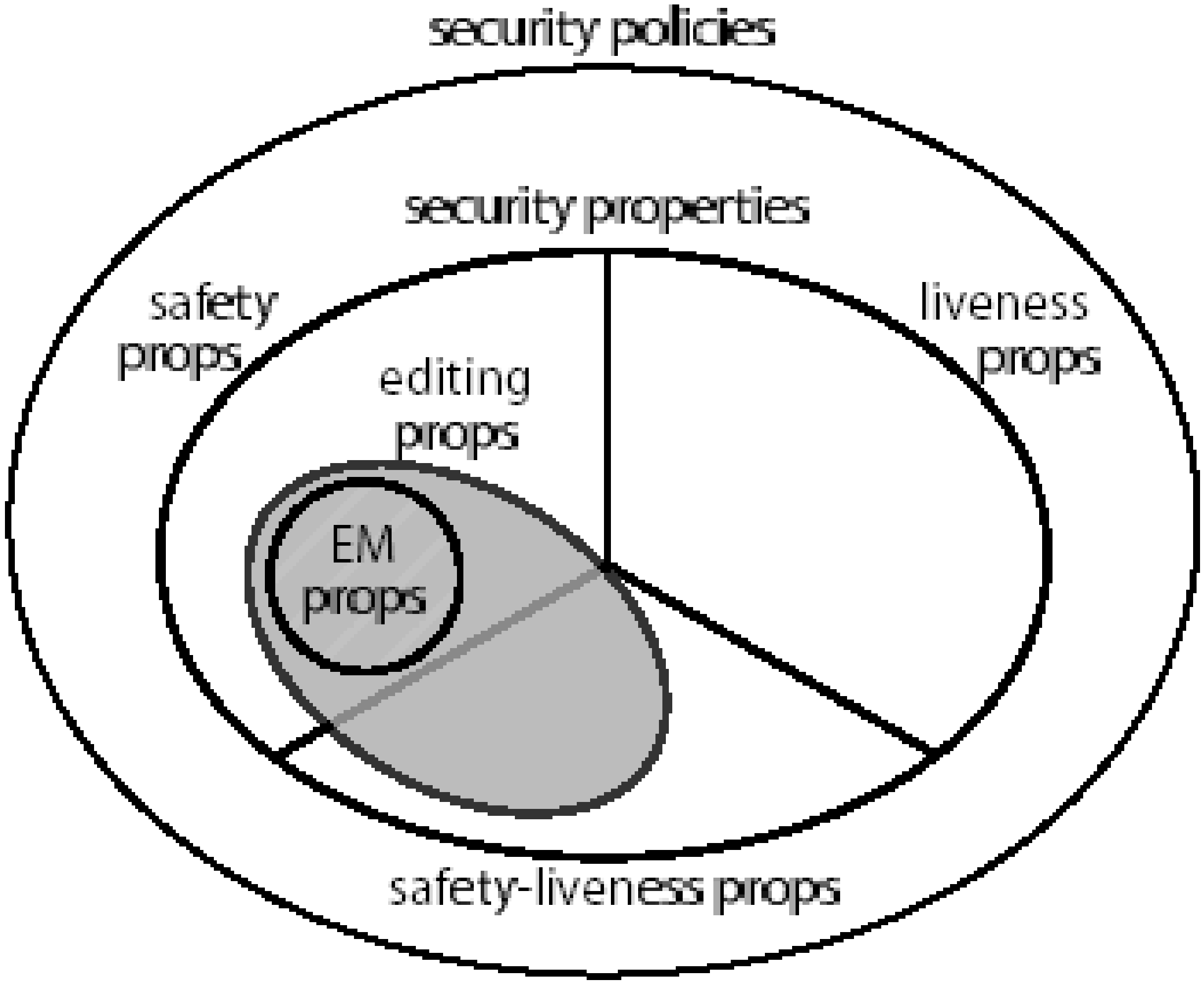


Each operation of A generates an input into security automaton of monitor

If monitor can make transition, operation of A is authorized.
If not, the monitor stops A before B sees the result.

Bauer: the automaton can edit the results

Enforceable Security Policies



Policy Enforcement by Static Program Analysis

- Check program at compile time whether it may contain security leaks at runtime.

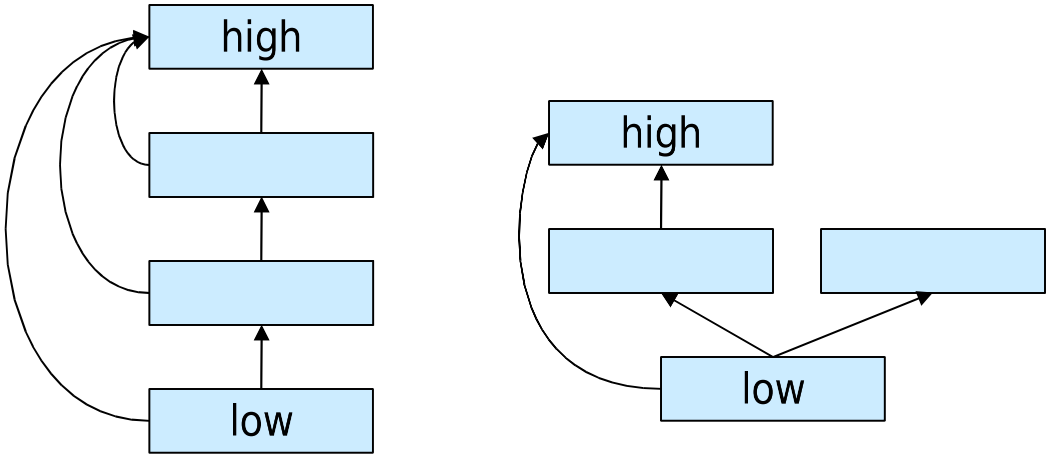
```
int low_observable;
int secret_key;

void foo() {

   if (c < 5)
      low_observable = secret_key;

}
```

Informaiton Flow

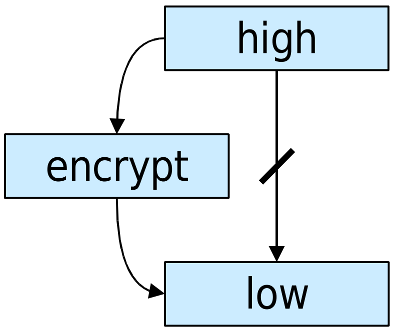
- Information Flow Policies
  - Bell La Padua ; Lattice Security ; Chinese Wall

(S : set_of[Label] ; dom : [Obj -> Label] ; ~/~> $\subseteq$ Label x Label)

transitive flow policies



intransitive flow policies



Information Flow

- Reasoning about several security policies
  - Confidentiality:
    - A ~/~> B =>
      B cannot deduce information on A (A’s data), A is confidential with respect to B
  - Integrity:
    - A ~/~> B =>
      B’s execution is independent of information / results from A, B is integer with respect to A
  - Availability:
    - A ~/~> B =>
      B’s availability is independent of information / results from A, B’s availability cannot be affected by A

Noninterference

- Intuitively:
  - a low classified observer cannot distinguish the outputs of a system that is presented an input that differs only in high variables
- Formally:
  - partial equivalence relation on states: s ~L s'
  - Noninterference:

$$s \sim_L s' \Rightarrow [[p]](s) \sim_L [[p]](s')$$

Examples: Confidentiality of Programs

```
int l {low};        variable that is externally observable after program terminates
int h {high};       variable storing confidential data

void foo() {
      l = h;
}

void bar() {
      if (h % 2)==1 {
            l = 1;
      }
}

void sec() {
      if (h % 2)==1 {
            h = h + 4;
      }
}
```

```
void long_op() {
   if (h % 2) == 1 {
 while (int i < 10000) { i++; }
   }
}

void terminate() {
   if (h%2) == 1 {
      while (true);
   }
}
```

Secure Type Systems

- Program is noninterference secure if it is typeable
  - Notation:
    - |- exp : t expression has type t according to typing rules
    - [pc] |- C programm C is typeable in security context [pc]

- Security Type Systems with Static Types
  - Typing rules for a simple while language

$$[\mathrm{E1\text{–}2}] \quad \vdash exp : high \qquad \frac{h \notin Vars(exp)}{\vdash exp : low}$$

$$[\mathrm{C1\text{–}3}] \quad [pc] \vdash \mathsf{skip} \qquad [pc] \vdash h := exp \qquad \frac{\vdash exp : low}{[low] \vdash l := exp}$$

$$[\mathrm{C4\text{–}5}] \quad \frac{[pc] \vdash C_1 \quad [pc] \vdash C_2}{[pc] \vdash C_1 ; C_2} \qquad \frac{\vdash exp : pc \qquad [pc] \vdash C}{[pc] \vdash \mathsf{while}\ exp\ \mathsf{do}\ C}$$

$$[\mathrm{C6\text{–}7}] \quad \frac{\vdash exp : pc \quad [pc] \vdash C_1 \quad [pc] \vdash C_2}{[pc] \vdash \mathsf{if}\ exp\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2} \qquad \frac{[high] \vdash C}{[low] \vdash C}$$

Secure Type Systems

[E1–2] $\vdash exp : high$ $\qquad \dfrac{h \notin Vars(exp)}{\vdash exp : low}$

[C1–3] $[pc] \vdash \mathsf{skip}$ $\qquad [pc] \vdash h := exp$ $\qquad \dfrac{\vdash exp : low}{[low] \vdash l := exp}$

[C4–5] $\dfrac{[pc] \vdash C_1 \quad [pc] \vdash C_2}{[pc] \vdash C_1 ; C_2}$ $\qquad \dfrac{\vdash exp : pc \quad [pc] \vdash C}{[pc] \vdash \mathsf{while}\ exp\ \mathsf{do}\ C}$

[C6–7] $\dfrac{\vdash exp : pc \quad [pc] \vdash C_1 \quad [pc] \vdash C_2}{[pc] \vdash \mathsf{if}\ exp\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2}$ $\qquad \dfrac{[high] \vdash C}{[low] \vdash C}$

[low?] |- l := h;

C3 => |- h : low

E2 => h $\notin$ Vars(h)

l := 0;

C3 => |- 0 : low

E2 => h $\notin$ Vars(0)

Secure Type Systems

[E1–2] $\vdash exp : high$ $\qquad \dfrac{h \notin Vars(exp)}{\vdash exp : low}$

[C1–3] $[pc] \vdash \text{skip}$ $\qquad [pc] \vdash h := exp$ $\qquad \dfrac{\vdash exp : low}{[low] \vdash l := exp}$

[C4–5] $\dfrac{[pc] \vdash C_1 \quad [pc] \vdash C_2}{[pc] \vdash C_1 ; C_2}$ $\qquad \dfrac{\vdash exp : pc \quad [pc] \vdash C}{[pc] \vdash \text{while } exp \text{ do } C}$

[C6–7] $\dfrac{\vdash exp : pc \quad [pc] \vdash C_1 \quad [pc] \vdash C_2}{[pc] \vdash \text{if } exp \text{ then } C_1 \text{ else } C_2}$ $\qquad \dfrac{[high] \vdash C}{[low] \vdash C}$

[low?] |- l := h;

C3 => |- h : low

E2 => h ~~$\notin$ Vars~~(h)

l := 0;

C3 => |- 0 : low

E2 => h $\notin$ Vars(0)

Secure Type Systems

- Flow Sensitive Security Type Systems

[low?] |- l := h; l := 0;

s0	h	l := h ;	0	l := 0
l : L	L	H	H	L
h : H	H	H	H	H
res	H		L	

check for decreasingness

Questions

- References
  - Matt Bishop:
    Computer Security – Art and Science
  - P. Gallagher:
    A Guide to Understanding Covert Channel Analysis of Trusted Systems [TCSEC]
  - Proctor, Neumann:
    Architectural Implications of Covert Channels
  - Kemmerer, Porras:
    Covert Flow Trees: A visual approach to detecting covert storage channels
  - Sabelfeld, Myers:
    Language-based information-flow security
  - *Walker, Bauer, Ligatti:*
    More enforcable security policies